UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

**Algoritmos Genéticos**

**Proyecto 3**

Aprendizaje

Profesor: M.C. Hugo E. Estrada León

Alumnos:

Sánchez González Julio

Semestre 2019-1

Fecha de entrega: 14 de noviembre de 2018

**Objetivo**

El alumno programará un algoritmo genético con la finalidad de maximizar una función, el alumno conocerá los efectos de la tasa de cruza y mutación. También aplicará algoritmos de selección de individuos para desarrollar el programa el cual elija a los mejores individuos para un determinado problema y hacer la elección de los mejores parámetros para el problema.

**Introducción**

Algoritmos Genéticos

El algoritmo genético es una técnica de búsqueda basada en la teoría de la evolución de Darwin. En los últimos años, la comunidad científica internacional ha mostrado un creciente interés en una nueva técnica de búsqueda basada en la teoría de la evolución y que se conoce como el algoritmo genético. Esta técnica se basa en los mecanismos de selección que utiliza la naturaleza, con base a+ individuos más aptos de una población, que son los que sobreviven, al adaptarse más fácilmente a los cambios que se producen en su entorno.

Un investigador de la Universidad de Michigan llamado John Holland logró que las computadoras aprendieran por sí mismas. A la técnica que inventó Holland se le llamó originalmente "planes reproductivos", pero se hizo popular bajo el nombre "algoritmo genético" tras la publicación de su libro en 1975.

Los Algoritmos Genéticos (AGs) son métodos adaptativos que pueden usarse para resolver problemas de búsqueda y optimización. Están basados en el proceso genético de los organismos vivos. A lo largo de las generaciones, las poblaciones evolucionan en la naturaleza de acorde con los principios de la selección natural y la supervivencia de los más fuertes, postulados por Darwin. Por imitación de este proceso, los Algoritmos Genéticos son capaces de ir creando soluciones para problemas del mundo real. Un algoritmo genético consiste en una función matemática o una rutina de software que toma como entradas a los ejemplares y retorna como salidas cuales de ellos deben generar descendencia para la nueva generación.

Versiones más complejas de algoritmos genéticos generan un ciclo iterativo que directamente toma a la especie (el total de los ejemplares) y crea una nueva generación que reemplaza a la antigua una cantidad de veces determinada por su propio diseño. Una de sus características principales es la de ir perfeccionando su propia heurística en el proceso de ejecución, por lo que no requiere largos períodos de entrenamiento especializado por parte del ser humano, principal defecto de otros métodos para solucionar problemas, como los Sistemas Expertos.

**Desarrollo**

**Programa 1**

Para el programa 1 se pidió usar un algoritmo genético para maximizar una función que consistía en identificar el monto más beneficioso en 4 tiendas para sacar la máxima ganacia.

Se comenzó definiendo algunos parámetros por default que se necesitaban al momento del entrenamiento:

pob = 50

gananciaT1 = [0.00, 0.28, 0.45, 0.65, 0.78, 0.90, 1.02, 1.13, 1.23, 1.32, 1.38]

gananciaT2 = [0.00, 0.25, 0.41, 0.55, 0.65, 0.75, 0.80, 0.85, 0.88, 0.90, 0.90]

gananciaT3 = [0.00, 0.15, 0.25, 0.40, 0.50, 0.62, 0.73, 0.82, 0.90, 0.96, 1.00]

gananciaT4 = [0.00, 0.20, 0.33, 0.42, 0.48, 0.53, 0.56, 0.58, 0.60, 0.60, 0.60]

Como métodos más utilizados hicimos los siguientes:

* Función de Aptitud

def fAptitud(T1, T2, T3, T4):

v = abs((T1 + T2 + T3 + T4) - 10)

aptitud = ((T1 + T2 + T3 + T4) / ((500 \* v) + 1))

return aptitud

* Creación de Individuos:

def creacionIndividuos():

for i in range(pob):

individuos.append([random.randint(0,10),random.randint(0,10),random.randint(0,10),random.randint(0,10)])

* Cálculo de Aptitud

def calculoAptitudes():

sumaApt = 0

T = 0

for individuo in individuos:

aptitudes.append(fAptitud(gananciaT1[individuo[0]], gananciaT2[individuo[1]], gananciaT3[individuo[2]], gananciaT4[individuo[3]]))

for aptitud in aptitudes:

sumaApt += aptitud

f = sumaApt/pob

for aptitud in aptitudes:

vei = aptitud/f

valoresEsperados.append(vei)

T += vei

* Selección por Ruleta

def ruleta():

nSel = 0

while pob > nSel:

r = random.uniform(0, T)

sumaVE = 0

for n, vei in enumerate(valoresEsperados):

sumaVE += vei

if sumaVE >= r:

nSel += 1

seleccionados.append(individuos[n])

* Cuza a 2 puntos:

def cruza(tasa\_cruza = 0.8):

aMutar = []

for i in range(pob):

r = random.uniform(0,1)

if r < tasa\_cruza:

aMutar.append(i)

for n, iPadre in enumerate(aMutar):

if iPadre == aMutar[-1]:

seleccionados[iPadre] = cxTwoPoint(seleccionados[iPadre], seleccionados[aMutar[0]])[0]

else:

seleccionados[iPadre] = cxTwoPoint(seleccionados[iPadre], seleccionados[aMutar[n + 1]])[0]

* Mutación por Intercambio Reciproco

def mutacion(tasa\_mutacion = 0.01):

for n, individuo in enumerate(seleccionados):

seleccionados[n] = mutUniformInt(individuo, 0, 10, tasa\_mutacion)[0]

Por último, se realizó un llamado de las funciones 20 veces para simular las generaciones

creacionIndividuos()

print(individuosIni)

valoresEsperados = calculoAptitudes(individuosIni)

nuevosSeleccionados = ruleta(individuosIni, valoresEsperados)

nuevaCruza = cruza(nuevosSeleccionados)

nuevaMutacion = mutacion(nuevaCruza)

for i in range(20):

print(nuevaMutacion)

valoresEsperados = calculoAptitudes(nuevaMutacion)

nuevosSeleccionados = ruleta(nuevaMutacion, valoresEsperados)

print("Nuevos seleccionados")

print(nuevosSeleccionados)

nuevosCruza = cruza(nuevosSeleccionados)

nuevaMutacion = mutacion(nuevaCruza)

print("Últimos más aptos: ")

print(nuevaMutacion)

**Resultados**

Se notó que para

**Conclusiones**

Sánchez González Julio

El algoritmo backpropagation es eficiente en el entrenamiento de una máquina con una o varias capas para representar una función que pueda ser capaz de clasificar entradas de las categorías dadas como entrenamiento o aquellas que tiene características parecidas a alguna de éstas. Esto lo hace un algoritmo adaptable a las entradas que reciba con base a un entrenamiento con datos y la posibilidad de sumar funciones de activación para ofrecer el mejor resultado.

En cuanto al perceptrón para el reconocedor de caracteres podemos decir que es eficiente para datos binarios o de dos categorías, ya que es una solo neurona, pero al momento de implementarlo en el reconocedor de caracteres y usar más de una capa, se hace sencillo poder entrenar a la máquina con un conjunto de datos para entrenarla y obtener el resultado, no óptimo, pero funciona para lo que se requiere.

Cano Olguin Luis Sergio

Como tal el desarrollo de este proyecto nos mostró la utilidad que pueden llegar el algoritmo en cuanto a la aproximación de funciones y al entrenamiento de maquina, tambien pudimos ver como la cantidad de neuronas en la capa oculta puede hacerlo más eficiente, junto con la propia cantidad de capas, aunque también aumenta la posibilidad de tener sobre entrenamiento, ya que el algoritmo no converge el cómo determinamos cuándo detenernos define que tan bueno fue el entrenamiento, el cual también es enormemente afectado por la tasa de aprendizaje, con la cual determinamos qué tan rápido nos movemos, si es demasiado grande podemos terminar más rápido, pero es más probable que nos alejemos del resultado, mientras que con una tasa baja tal vez tardemos un poco más pero nos acercaremos de manera más segura al resultado óptimo.

**Bibliografía**

* <http://disi.unal.edu.co/~lctorress/RedNeu/LiRna004.pdf>
* <https://medium.com/datathings/neural-networks-and-backpropagation-explained-in-a-simple-way-f540a3611f5e>
* <http://www.lcc.uma.es/~munozp/documentos/modelos_computacionales/temas/Tema4MC-05.pdf>
* <https://es.wikipedia.org/wiki/Perceptr%C3%B3n_multicapa>

**Manual para los Programas**

Programa 1

Se instalan las siguientes librerías:

* python -m pip install -U pip
* python -m pip install -U matplotlib
* pip install numpy

Para correr lo solamente hay que iniciar el programa con el comando “python Ejercicio1.py” en consola y si se quiere cambiar los datos se pueden cambiar desde la declaración de éstos en la siguiente parte del código:

w1 = np.array([[-0.27],[-0.41]])

b1 = np.array([[-0.48],[-0.13]])

w2 = np.array([[0.09,-0.17]])

b2 = 0.48

Para mandar a llamar al algoritmo, éste tiene que recibir los pesos y bias, la iteración actual y las totales.

def algoritmo(p,w1,b1,w2,b2,i):

Programa 2

Se instalan las mismas librerías que el programa 1.

Para correr lo solamente hay que iniciar el programa con el comando “python Ejercicio2.1.py” en consola y si se quiere cambiar los datos se pueden cambiar desde la declaración de éstos en la siguiente parte del código:

def algoritmo(entradas,neuronas,rate):

    w1 = np.random.rand(neuronas,entradas)-0.5

    b1 = np.random.rand(neuronas,entradas)-0.5

    w2 = np.random.rand(4,neuronas)-0.5

    b2 = np.random.rand(4,entradas)-0.5

    w3 = np.random.rand(entradas,4)-0.5

    b3 = 0.48

Reconocedor de caracteres:

Se instalan las misma librerías que el programa 1 y 2.

Para correr lo solamente hay iniciar el programa con el comando “python Reconocedor.py” en consola y si se quieren cambiar los datos hay que modificar el archivo llamado “Buscar.txt” el cual es leído por el programa para reconocer un carácter.

archivo = open("Buscar.txt")

x = []

for line in archivo.read():

    y=[v for v in line.split()]

    if y!=[ ]:

        x.append(int(y[0]));

        #############

O si se desea se podría subir otro archivo con el formato siguiente:

0 1 1 0 1 0 0 0 0 0 0 0 1 0 1 0